ECE 411 Lab Sheet (Fall 2024)

Instructor: Dr. Chau-Wai Wong

Note: The lab exercises will be assigned in a future HW as bonus problems. If you want, you may start working on them before they are assigned as HW problems.

Topics Covered:

1. Python Package Management with Conda Environment

Different machine learning projects, depending on when they were developed, may work only with a specific version of Python and a unique set of dependent Python packages. Miniconda ("conda" hereafter; please follow this quick command line instructions to install conda if it is not already installed) allows creating multiple Python environments and will leave your operating system untouched. A conda environment of an open-source project can be easily recreated on your own machine with a few lines of commands with the help of a list of packages commonly specified in `requirements.txt` or `.yml`.

Conda suits ML researchers and engineers because they often have to run others' open-source projects. Conda can be an efficient tool to deal with badly documented projects whose Python and/or package versions are unspecified. A common mitigation is to guess (with the help of LLMs) the versions of Python and their packages from their release and updated time stamps of the codebase. In this case, it will be handy to create a new conda-based Python environment with improved guesses if the previous conda environment prevent code from running successfully.

Conda is the standard approach for working on and releasing serious ML projects in both academia and industry. Colab or Jupyter Notebook, which is easier to learn and visually accommodating, are better for demo purposes.

Conda creation

Let's use the installation of the ISLR package within a conda environment as an example. Please follow the instructions below even though they were originally adapted from ISLP's official installation instructions.

For Windows users, please install a Windows Subsystem for Linux (WSL) following Microsoft's official instructions. Once the installation completes, you will be able to work with the subsystem as if you are working on a Linux machine. Note that ISLP in their official installation instructions also provided a Windows-native Anaconda approach. You are discouraged to try this route as ML is almost always done in Linux.

To create a conda environment in a Linux or Mac environment, you may run:

```
conda create -n islp python=3.10
```

Here, parameters `-n islp` specifies the name of the conda environment. Keeping the name short will help as you will type it often. Keeping the name informative will help when you decide which conda environment to activate among 10+ environments on your computer.

Parameter `python=3.10` specifies the version of the Python interpreter we will install. For ML purposes, it's a good practice to specify a version that is neither too old or too new. A too new version may not be supported yet by key ML packages such as `pytorch`, whereas a too old version may lead to the installation of older dependency libraries that are out of maintenance and/or do not run on your new GPU card.

To completely remove a conda environment named `env_name` (no matter how many packages you installed), you may use:

```
conda env remove -n env_name
```

To list existing conda environments, you may use:

```
conda info --envs
```

To list all Python packages installed under the `env_name` conda environment, you may use:

```
conda list -n env_name
```

To run Python code in the `islp` environment, you must activate it:

```
conda activate islp
```

Once the `islp` environment is activated, the command prompt will be prefixed accordingly, e.g.,

> (islp) username@computer_name:~$

Once a conda environment is successfully created and activated, we can start installing Python packages using the `pip install package_name` or `conda install package_name` command. In this tutorial, we will focus on the former approach as it mirrors the non-conda Python package installation process. Before moving forward, please use the `which pip` (or `which pip3`) within the activated conda environment to double-check whether the `pip` (or `pip3`) program being called locates within conda. You should see outputs similar like `/home/username/miniconda3/envs/islp/bin/pip` rather than `/usr/bin/pip`. If the latter is displayed, please Google to figure out how to install or activate the conda environment correctly. Once this step is complete, list all Python packages installed using a command given above and put a screenshot in your report.

Package installation

Since the Python packages have interdependences, my rule-of-thumb is to install "complex" packages that depends on GPUs and have more dependencies. For example, `pytorch` and `tensorflow` needs to be installed first because the dependency packages developed at the 1080Ti GPU era were very different from the those developed at the 4090 GPU era. Installing `numpy` immediately after a conda environment creation will likely retrieve the latest version of `numpy` and fail the subsequent installation of `pytorch` or `tensorflow` for legacy GPU or codebase.

For "complex" packages, we should stick to official installation guides:

1. Install PyTorch (for modern ML projects)
2. Install Tensorflow (mainly for legacy projects)

Both installation guides contain validation code segments for you to ensure that your installation is successful. This is extremely important for those of you who use GPU for ML projects. If the validation code does not provide intended output, you may create another conda and try again. Once you get one conda environment working, you may delete the earlier conda environment(s).

To determine which core ML package, i.e., PyTorch or Tensorflow, should be installed for running an open-source project, you may check their requirements.txt (this is the ISLP's example). Take a screenshot of all installed packages up to this stage and put it in the report.

Once the "complex" packages such as PyTorch and Tensorflow are correctly installed in your conda environment, you may start installing other Python packages. For example, to install the `ISLP` package, you may run `pip install ISLP`. In case of a big open-source project, its authors may provide a `requirements.txt` file that include the names of all necessary packages. You may install all of them in one shot by providing the `-r` parameter and concatenating file path to the `requirements.txt`, e.g.,

```
pip install -r https://raw.githubusercontent.com/intro-stat-
learning/ISLP_labs/v2/requirements.txt
```

If certain packages in `requirements.txt` causes problems when being installed to the current conda environment, you may download `requirements.txt`, remove these packages, save `requirements.txt`, and install the remaining packages by passing the file with the new path `~/Downloads/requirements.txt` as the last parameter to the command above. Take a screenshot of all installed packages and put it in the report.

2. VS Code Remote Development

You can do remote coding and even real-time debugging using VS Code's Remote Development extension.

Download a private key named `id_rsa_student` for accessing Dr. Wong's remote GPU server. To download the key, you need to log into Piazza first.

To install VS Code and its Remote Development extension, follow the general steps in this YouTube tutorial video. You can skip the public-private key creation part as you have been provided the private key and Dr. Wong has already pre-deployed the public key on the GPU server. Make sure your Remote Development `config` file is has the following content block:

```
Host Ece411StudentGpuServer
    HostName tlsa-service.ddns.net
    User ece411student
    Port 2222
    IdentityFile C:/Users/username/Downloads/id_rsa_student
```

The first 4 lines should be exactly the same as provided, and the 5th line should be dependent on exact file path of private key file on your computer. Take a screenshot of the `config` file and put it in your report.

Once you succesfully connect to the remote GPU server, a blue "Open Folder" button will be displayed. Once you click it, please follow the instruction to open the `/home/ece411student/` folder. Next, on the top menu bar, click "Terminal" -> "New Terminal", and you will see a remote prompt is displayed: `ece411student@wong-home-linux:~$`. Type `nvidia-smi` and press enter to display the current GPUs' working conditions and utilization rates. Take a screenshot and put it in the report.

Again on the top menu bar, click "File" -> "New File", type in `firstname_lastname.py` (please fill in your name) to create an empty Python file. Put in the following content:

```
import numpy as np
print(np.random.rand(3, 2))
```

Click the start button (a triangle pointing to the right) on the top right corner of the text editor to run the code. If everything is correct, you should see the results appearing in the terminal generated by the code running on the remote GPU server. Take the final screenshot of the whole VS Code screen and put it in the report. Make sure the source code, Python file name, and the Python output are visible to get the full points.

3. High-Performance Computing (HPC) Training

Complete NC State HPC training and submit the certificate with your name. Your learning outcomes after completing this training are listed below.

Part 1 (Introduction to Basic HPC):

- know a basic subset of essential Linux commands and how to use a command line text editor
- understand and agree to the HPC Acceptable Use Policy
- log in to the Henry2 Linux cluster
- know how to find help and information on the HPC website

Part 2 (Basic Storage and File Transfer):

- know the various storage options and the properties for each, including proper use, how much space is available, backup policies, and login vs. compute nodes
- know how to transfer files, including the best way to transfer a single file, multiple files, and whether HPC supports file transfer using a GUI interface

Part 3 (Running Serial Jobs):

- understand the difference between a login node and compute node
- understand why environment variables need to be set before using an application
- understand the concept of scheduling jobs and how to submit a basic job
- understand how to debug and test code without running on a login node
- know how to find example batch scripts on the HPC website

Part 4 (Running Parallel Jobs):

- understand the difference between shared and distributed memory parallelism
- understand what nodes, processors, and cores are as they relate to submitting a job
- understand that Henry2 has different types of nodes, which is analogous to buying computers with different specs, and know how to find the specs of different nodes on the cluster

Extra resources: NC State HPC GPU-Related Tips by Adrian Chan.